Load everthing required in console

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas_profiling import ProfileReport # `pip install pandas-profiling`
```

In [2]:
```
# For dealing with xlsx format.
!pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\programdata\anaconda3\lib\site-packages (3.0.7)
Requirement already satisfied: et-xmlfile in c:\programdata\anaconda3\lib\site-packages (from openpyxl) (1.0.1)
```

Load airlines_dataset.xlsx data file using pandas.read_excel

Same for csv also

In [3]:
```
df = pd.read_excel("./airlines_dataset.xlsx")
```

Examine first 5 rows of data

In [4]:
```
df.head()
```

Out[4]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

Examine last 5 rows of data

In [5]:
```
df.tail()
```

Out[5]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

Return a tuple representing the dimensionality of the DataFrame.

(Row, Column)

In [6]:
```
df.shape
```

Out[6]:
```
(10683, 11)
```

Basic information of the dataset using .info()

In [7]:
```
df.info()


# Every column is string object except the `Price` is integer
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

Calculate summary statistics of all column `df.describe(include= 'all')`

default one without parameters gives summary statistics of integer column only `df.describe()`

In [8]:
```
df.describe(include='all')
```

Out[8]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
count	10683	10683	10683	10683	10682	10683	10683	10683	10682	10683	10683.000000
unique	12	44	5	6	128	222	1343	368	5	10	NaN
top	Jet Airways	18/05/2019	Delhi	Cochin	DEL → BOM → COK	18:55	19:00	2h 50m	1 stop	No info	NaN
freq	3849	504	4537	4537	2376	233	423	550	5625	8345	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9087.064121
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4611.359167
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1759.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5277.000000
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8372.000000
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12373.000000
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	79512.000000

default one without parameters gives summary statistics of integer column only `df.describe()`

In [9]:
```
df.describe()
```

Out[9]:

	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

Short introduction about dataset:

The dataset contains the `airline's data of Indian Airways from (2019)` which is verified in the below cells.

The dataset has `10683 rows & 11 columns` as of raw state.

`Columns` are destined to increase after encoding in pre-processing and `rows` are destined to decrease because of duplicate data.

`Airline` column consists of different names of airlines that flew in the given routes in the span of a year(2019).

`Date_of_Journey` indicates the date it flew from the `Destination` to the `Source` along the `Route` mentioned in the respective `columns` of the dataset.

The departure time `Dep_Time` is the time when the plane flew which contains the time object whereas the arrival time `Arrival_Time` contains time as well as the date if arrived the next day.

The `Total_Stops` column can be derived from the number of routes the plane flew.

There are only two `NaN` values and the `Additional_Info` column has no information in `{ > 50 % }` of the data. The column usually contains the data about the food serving, baggage details, and layover details.

The `Price` is the final column that indicates the price of service of airlines which can also be used as a `predictive variable`.

Airline

Categorical data

In [10]:
```
print('All Airlines: ', df.Airline.unique(), '\n')
# Names of every airlines

print('Total airlines count: ', len(df.Airline.unique()), '\n')
# Total unique airlines count

print(df.Airline.describe(), '\n')
# Description of the column consisting of most frequent airways

print(df.Airline.isnull().unique(), '\n')
# Checking if there are any null values in the column
```

```
print(df.Airline.value_counts())
# Each airline with the number of records it has
```

```
All Airlines:  ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir'
 'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'
 'Multiple carriers Premium economy' 'Trujet']

Total airlines count:  12

count          10683
unique            12
top       Jet Airways
freq            3849
Name: Airline, dtype: object

[False]

Jet Airways                          3849
IndiGo                               2053
Air India                            1752
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
```

Plot pie chart for Airline.value_counts()

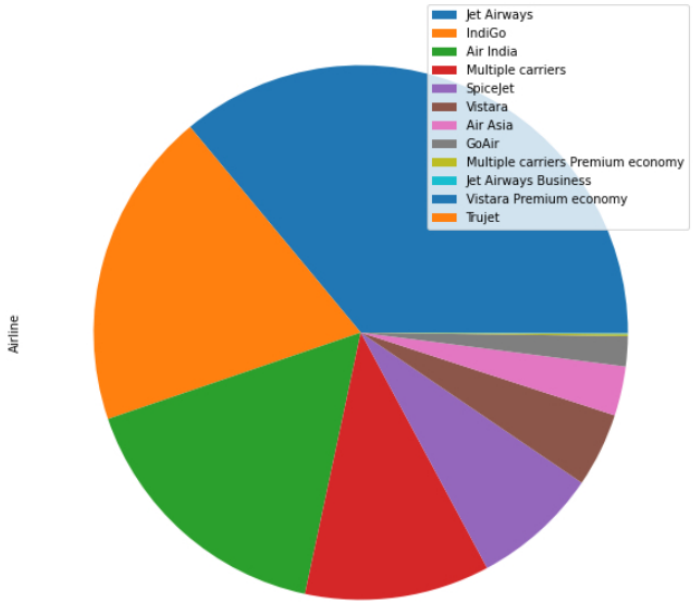
`figsize= (10, 10)` => figure size to be plot

`labeldistance= None` => removes labelling of each pie

`legend= True` => gives that beautiful labelled legend box with color

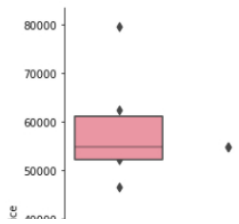
In [11]:
```
df.Airline.value_counts().plot.pie(figsize=(10, 10), labeldistance=None, legend=True)
```

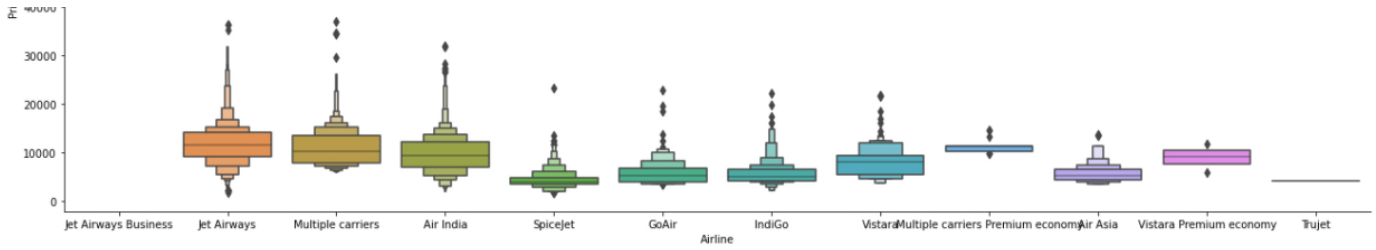
Out[11]:
```
<AxesSubplot:ylabel='Airline'>
```



In [12]:
```
# Plotting Price with respect to Airline
sns.catplot(y="Price", x="Airline", data=df.sort_values("Price", ascending=False), kind="boxen", height=6, aspect=3)
```

Out[12]:
```
<seaborn.axisgrid.FacetGrid at 0x1ea87fdc280>
```





Date_of_Journey

In [13]:
```
date = pd.to_datetime(df.Date_of_Journey)
# Converting Date_of_Journey object into a datetime64 object

date_only = date.dt.date
# Keeping the date part only as the time by default is 00:00:00

print(date_only.sort_values(ascending=True).iloc[[0, 1, df.shape[0]/2, -2, -1]], '\n')
# Printing top 2, middle and bottom 2 rows
```

```
959     2019-01-03
6336    2019-01-03
3292    2019-05-24
4997    2019-12-06
1081    2019-12-06
Name: Date_of_Journey, dtype: object
```

In [14]:
```
print("Last date is: ", date_only.max())
print("Earliest date is: ", date_only.min())

# The lower bound and the upper bound of data are January 3 to December 6, 2019.
```

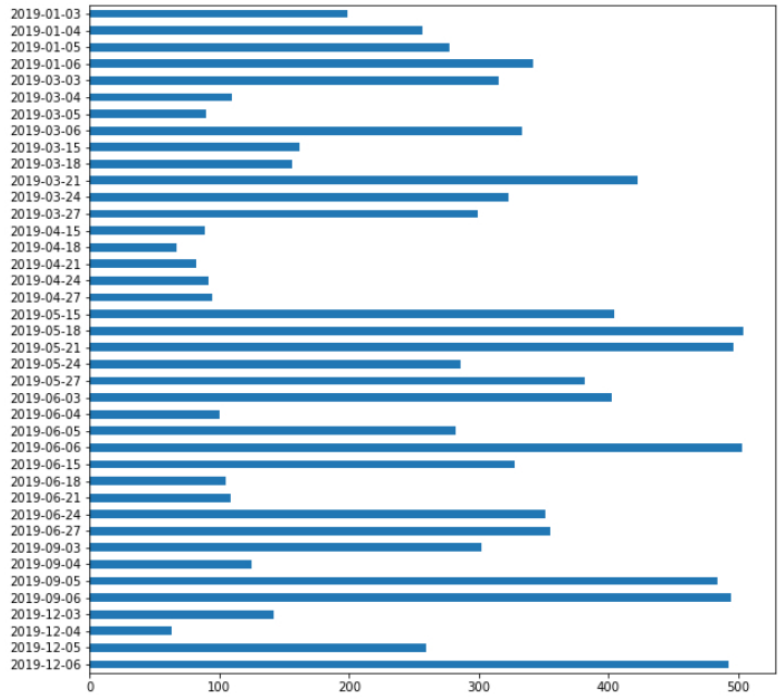
```
Last date is:  2019-12-06
Earliest date is:  2019-01-03
```

In [15]:
```
print(date_only.value_counts(dropna=False).sort_index())

# Finding total flights in each day of the year
```

```
2019-01-03    199
2019-01-04    257
2019-01-05    277
2019-01-06    342
2019-03-03    315
2019-03-04    110
2019-03-05     90
2019-03-06    333
2019-03-15    162
2019-03-18    156
2019-03-21    423
2019-03-24    323
2019-03-27    299
2019-04-15     89
2019-04-18     67
2019-04-21     82
2019-04-24     92
2019-04-27     94
2019-05-15    405
2019-05-18    504
2019-05-21    497
2019-05-24    286
2019-05-27    382
2019-06-03    403
2019-06-04    100
2019-06-05    282
2019-06-06    503
2019-06-15    328
2019-06-18    105
2019-06-21    109
2019-06-24    351
2019-06-27    355
2019-09-03    302
2019-09-04    125
2019-09-05    484
2019-09-06    495
2019-12-03    142
2019-12-04     63
2019-12-05    259
2019-12-06    493
Name: Date_of_Journey, dtype: int64
```

In [16]:
```
date_only.value_counts().sort_index(ascending=False).plot(kind='barh', figsize=(10,10))

# Most flights were on May 18 (Saturday), June 6 (Thursday), September 6 (Friday) and December 6 (Friday).
```

Out[16]:
```
<AxesSubplot:>
```



In [17]:
```
# Since the data consists of the year 2019 and it is repeated throughout the column, the year can be removed.


# Renaming the column
df.rename(columns={"Date_of_Journey": "Date_of_Journey_2019"}, inplace= True)
```

In [18]:
```
# Removing the redundant date (upto -5 slice => 2019/)
df.Date_of_Journey_2019 = df.Date_of_Journey_2019.str.slice(stop=-5)
```

In [19]:
```
df
```

Out[19]:

	Airline	Date_of_Journey_2019	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	9/04	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

10683 rows × 11 columns

In [20]:
```
# Separating Journey Day
df['Journey_Day'] = pd.to_datetime(df.Date_of_Journey_2019, format="%d/%m").dt.day
```

In [21]:
```
# Separating Journey Month
df['Journey_Month'] = pd.to_datetime(df.Date_of_Journey_2019, format="%d/%m").dt.month
```

In [22]:
```
df
```

Out[22]:

	Airline	Date_of_Journey_2019	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_Day	Journey_Month
0	IndiGo	24/03	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	1/05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	9/06	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	12/05	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
					BLR → NAG →								

4	IndiGo	01/03	Banglore	New Delhi	DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	9/04	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107	9	4
10679	Air India	27/04	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145	27	4
10680	Jet Airways	27/04	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229	27	4
10681	Vistara	01/03	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648	1	3
10682	Air India	9/05	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753	9	5

10683 rows × 13 columns

In [23]:
```
# dropping journey column
df.drop(columns=['Date_of_Journey_2019'], axis=1, inplace=True)
```

In [24]:
```
df
```

Out[24]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_Day	Journey_Month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3
...	...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107	9	4
10679	Air India	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145	27	4
10680	Jet Airways	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229	27	4
10681	Vistara	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648	1	3
10682	Air India	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753	9	5

10683 rows × 12 columns

Source and Destination

In [25]:
```
print(df.Source.unique(), '\n')
# Unique sources for airlines departure

print(df.Source.value_counts(), '\n')
# Departure source count from most to least

print(df.Source.isnull().unique(), '\n')
# Checking if there are NaN values in source

#----------------------------------------

print(df.Destination.unique(), '\n')
# Unique destinations for airlines arrival

print(df.Destination.value_counts(), '\n')
# Arrival destination count from most to least

print(df.Destination.isnull().unique(), '\n')
# Checking if there are NaN values in destination
```

```
['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai'] 

Delhi       4537
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64 

[False] 

['New Delhi' 'Banglore' 'Cochin' 'Kolkata' 'Delhi' 'Hyderabad'] 

Cochin       4537
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64 
```

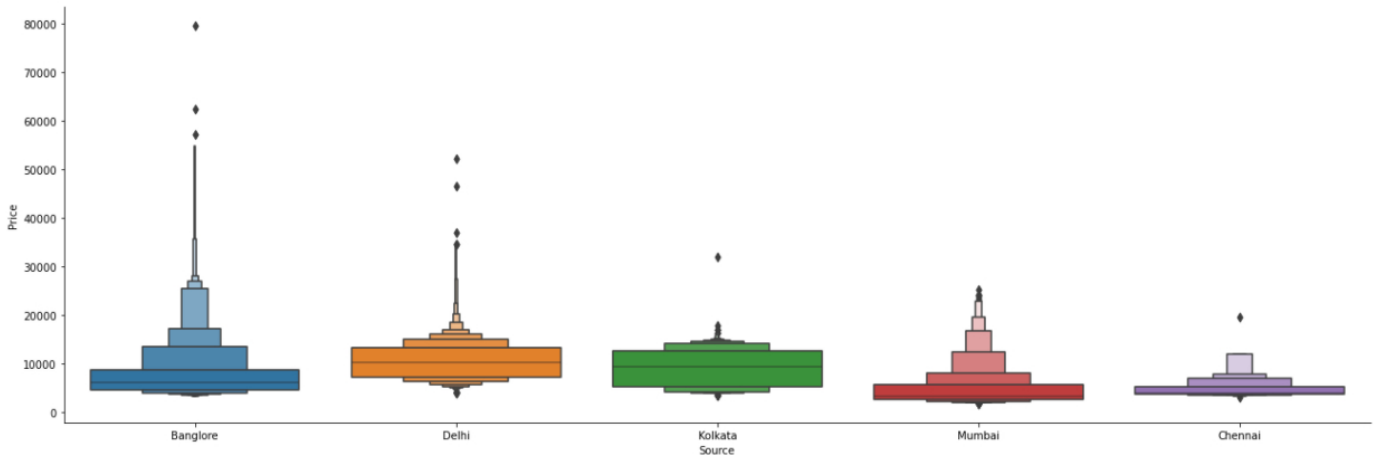
```
[False]
```

In [26]:
```
delhi_count = (df.Destination.values == ['Delhi']).sum()
print("Delhi count: ", delhi_count, '\n')
# Total instances of delhi data in destination

new_delhi_count = (df.Destination.values == ['New Delhi']).sum()
print("New Delhi count: ", new_delhi_count, '\n')
# Total instances of new delhi data in destination

# Total delhi instances
print("Total delhi count: ", delhi_count + new_delhi_count)
# Delhi would still come to the third to the most visited destinations after Cochin and Banglore.
```

```
Delhi count:  1265

New Delhi count:  932

Total delhi count:  2197
```

In [27]:
```
# Plotting price with respect to source
sns.catplot(y="Price", x="Source", data=df.sort_values("Price", ascending=False), kind="boxen", height=6, aspect=3)
```

Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x1ea887c7f10>
```



Routes and Stops

In [28]:
```
len(df.Route.unique())
# All the plane flew in 129 unique routes
```

Out[28]:
```
129
```

In [29]:
```
df.Route[0:10]
# Checking 10 instances of routes
```

Out[29]:
```
0                 BLR → DEL
1       CCU → IXR → BBI → BLR
2       DEL → LKO → BOM → COK
3             CCU → NAG → BLR
4             BLR → NAG → DEL
5                 CCU → BLR
6             BLR → BOM → DEL
7             BLR → BOM → DEL
8             BLR → BOM → DEL
9             DEL → BOM → COK
Name: Route, dtype: object
```

In [30]:
```
df.Total_Stops[0:10]
# Checking 10 instances of total stops
```

Out[30]:
```
0    non-stop
1     2 stops
2     2 stops
3      1 stop
4      1 stop
5    non-stop
6      1 stop
7      1 stop
8      1 stop
9      1 stop
Name: Total_Stops, dtype: object
```

In [31]:
```
print(df.Route.isnull().unique(), '\n')
```

```
print(df.Route.isnull().unique(), '\n')
# Checking if there are 'NaN' values in route

# There ARE 'NaN' values!!!
```

```
[False  True]
```

In [32]:
```
print(df.Total_Stops.isnull().unique(), '\n')
# Checking if there are NaN values in steps

# There ARE NaN values!!!
```

```
[False  True]
```

In [33]:
```
# Replacing stops string values with an integer

df.Total_Stops.replace(to_replace=['non-stop', '1 stop', '2 stops', '3 stops', '4 stops'], value=[0,1,2,3,4], inplace=True)
```

In [34]:
```
stops = df.Total_Stops
stops.unique()
```

Out[34]:
```
array([ 0.,  2.,  1.,  3., nan,  4.])
```

In [35]:
```
df.head()
```

Out[35]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_Day	Journey_Month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	0.0	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2.0	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2.0	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1.0	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1.0	No info	13302	1	3

In [36]:
```
# Counting the nan values

stops.isnull().value_counts()
#only one 'NaN' value
```

Out[36]:
```
False    10682
True         1
Name: Total_Stops, dtype: int64
```

In [37]:
```
# Finding the index of nan values

print(np.argwhere(np.isnan(np.array(stops))))
```

```
[[9039]]
```

In [38]:
```
df.iloc[9039]
# 'NaN' value only in 'Route' & 'Total_Stops'
```

Out[38]:
```
Airline               Air India
Source                    Delhi
Destination              Cochin
Route                       NaN
Dep_Time                  09:45
Arrival_Time       09:25 07 May
Duration               23h 40m
Total_Stops                NaN
Additional_Info        No info
Price                     7480
Journey_Day                  6
Journey_Month                5
Name: 9039, dtype: object
```

In [39]:
```
# Finding the most common 'Route' and 'Total_Stops' on the same airline with same sources, destinations and durations.

df[['Route', 'Total_Stops']].where((df.Source=='Delhi') & (df.Destination=='Cochin') & (df.Airline=='Air India') & (df.Duration=='23h
40m')).dropna().set_index('Route').value_counts()

# Since the most common route is 'DEL → HYD → MAA → COK' with 'Total_Stops=2', the values shall be replaced now.
```

Out[39]:
```
Total_Stops
2.0            14
1.0            12
dtype: int64
```

In [40]:
```
# Filling the values of routes
```

```
df.iloc[9039, 3] = 'DEL → HYD → MAA → COK'

# Filling the value of stops
df.iloc[9039, 7] = 2
```

In [41]:

```
# The value is replaced and no other nan values exist.

print(df.iloc[9039], '\n')

print(df.Total_Stops.isnull().value_counts())
```

```
Airline                      Air India
Source                           Delhi
Destination                     Cochin
Route              DEL → HYD → MAA → COK
Dep_Time                         09:45
Arrival_Time                09:25 07 May
Duration                       23h 40m
Total_Stops                        2.0
Additional_Info                No info
Price                             7480
Journey_Day                          6
Journey_Month                        5
Name: 9039, dtype: object

False    10683
Name: Total_Stops, dtype: int64
```

In [42]:

```
# Converting the float values to integers

df.Total_Stops = df.Total_Stops.apply(np.int64)
df.Total_Stops
```

Out[42]:

```
0        0
1        2
2        2
3        1
4        1
       ..
10678    0
10679    0
10680    0
10681    0
10682    2
Name: Total_Stops, Length: 10683, dtype: int64
```

In [43]:

```
df.head()
```

Out[43]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_Day	Journey_Month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	0	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1	No info	13302	1	3

Additional_Info

In [44]:

```
df.Additional_Info.value_counts(normalize=True).mul(100).round(8)
```

Out[44]:

```
No info                         78.114762
In-flight meal not included     18.552841
No check-in baggage included     2.995413
1 Long layover                   0.177853
Change airports                  0.065525
Business class                   0.037443
No Info                          0.028082
Red-eye flight                   0.009361
1 Short layover                  0.009361
2 Long layover                   0.009361
Name: Additional_Info, dtype: float64
```

Since 78% data in this column is empty or consists of "No info" which is of no use, it is safe to drop the column as the the data which do exist are not as too crucial.

In [45]:

```
df.drop(['Additional_Info'], axis = 1, inplace = True)
df.head(3)
```

Out[45]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Price	Journey_Day	Journey_Month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	0	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2	13882	9	6

Departure Time 'Dep_Time', 'Arrival_Time' and 'Duration'

In [46]:
```
# Checking if any nan values on departure time
df.Dep_Time.isna().unique()
```

Out[46]:
```
array([False])
```

In [47]:
```
# Formatting data into a more structured format
df['Dep_Hour'] = pd.to_datetime(df.Dep_Time).dt.hour

df['Dep_Min'] = pd.to_datetime(df.Dep_Time).dt.minute

# df.drop(['Dep_Time'], axis=1, inplace=True)

# dep_time =  pd.to_datetime(df.Dep_Time).dt.time
# print(dep_time)

# df.Dep_Time = dep_time
```

In [48]:
```
df.head()
```

Out[48]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	0	3897	24	3	22	20
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1	13302	1	3	16	50

df.Arrival_Time

In [49]:
```
df['Arrival_Hour'] = pd.to_datetime(df.Arrival_Time).dt.hour

df['Arrival_Min'] = pd.to_datetime(df.Arrival_Time).dt.minute

# df.drop(['Arrival_Time'],axis=1, inplace=True)
# arrival_time =  pd.to_datetime(df.Arrival_Time).dt.time
# df.Arrival_Time = arrival_time
```

In [50]:
```
df.tail()
```

Out[50]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min
10678	Air Asia	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	0	4107	9	4	19	55	22	25
10679	Air India	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	0	4145	27	4	20	45	23	20
10680	Jet Airways	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	0	7229	27	4	8	20	11	20
10681	Vistara	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	0	12648	1	3	11	30	14	10
10682	Air India	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2	11753	9	5	10	55	19	15

df.Duration

In [51]:
```
df["Dep_Time"]
```

Out[51]:
```
0        22:20
1        05:50
2        09:25
3        18:05
4        16:50
         ...
10678    19:55
10679    20:45
10680    08:20
10681    11:30
10682    10:55
Name: Dep_Time, Length: 10683, dtype: object
```

In [52]:
```
# Duration can be extracted like this.

dep_time =  pd.to_datetime(df.Dep_Time).dt.time
df.Dep_Time = dep_time
```

```
arrival_time =  pd.to_datetime(df.Arrival_Time).dt.time
df.Arrival_Time = arrival_time

from datetime import datetime, date

def time_between(df):
    return datetime.combine(date.today(), df["Arrival_Time"]) - datetime.combine(date.today(), df["Dep_Time"])

duration = df.apply(time_between, axis=1).astype('str')
duration
```

Out[52]:

```
0        -1 days +02:50:00
1          0 days 07:25:00
2        -1 days +19:00:00
3          0 days 05:25:00
4          0 days 04:45:00
              ...
10678      0 days 02:30:00
10679      0 days 02:35:00
10680      0 days 03:00:00
10681      0 days 02:40:00
10682      0 days 08:20:00
Length: 10683, dtype: object
```

In [53]:

```
df['Duration_hours']=duration.str.slice(start=-8, stop=-6)
df['Duration_mins']=duration.str.slice(start=-5, stop=-3)
```

In [54]:

```
df
```

Out[54]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min	Dura
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20:00	01:10:00	2h 50m	0	3897	24	3	22	20	1	10	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50:00	13:15:00	7h 25m	2	7662	1	5	5	50	13	15	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25:00	04:25:00	19h	2	13882	9	6	9	25	4	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05:00	23:30:00	5h 25m	1	6218	12	5	18	5	23	30	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50:00	21:35:00	4h 45m	1	13302	1	3	16	50	21	35	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10678	Air Asia	Kolkata	Banglore	CCU → BLR	19:55:00	22:25:00	2h 30m	0	4107	9	4	19	55	22	25	
10679	Air India	Kolkata	Banglore	CCU → BLR	20:45:00	23:20:00	2h 35m	0	4145	27	4	20	45	23	20	
10680	Jet Airways	Banglore	Delhi	BLR → DEL	08:20:00	11:20:00	3h	0	7229	27	4	8	20	11	20	
10681	Vistara	Banglore	New Delhi	BLR → DEL	11:30:00	14:10:00	2h 40m	0	12648	1	3	11	30	14	10	
10682	Air India	Delhi	Cochin	DEL → GOI → BOM → COK	10:55:00	19:15:00	8h 20m	2	11753	9	5	10	55	19	15	

10683 rows × 17 columns

Dropping redundant columns

In [55]:

```
# Dropping redundant columns
```

```
df.drop(columns=['Duration', 'Dep_Time', 'Arrival_Time'], inplace=True)
```

In [56]:
```
df
```

Out[56]:

	Airline	Source	Destination	Route	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min	Duration_hours	Duration_mins
0	IndiGo	Banglore	New Delhi	BLR → DEL	0	3897	24	3	22	20	1	10	02	50
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2	7662	1	5	5	50	13	15	07	25
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2	13882	9	6	9	25	4	25	19	00
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1	6218	12	5	18	5	23	30	05	25
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1	13302	1	3	16	50	21	35	04	45
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	Kolkata	Banglore	CCU → BLR	0	4107	9	4	19	55	22	25	02	30
10679	Air India	Kolkata	Banglore	CCU → BLR	0	4145	27	4	20	45	23	20	02	35
10680	Jet Airways	Banglore	Delhi	BLR → DEL	0	7229	27	4	8	20	11	20	03	00
10681	Vistara	Banglore	New Delhi	BLR → DEL	0	12648	1	3	11	30	14	10	02	40
10682	Air India	Delhi	Cochin	DEL → GOI → BOM → COK	2	11753	9	5	10	55	19	15	08	20

10683 rows × 14 columns

In [57]:
```
# Since route with the total_stops is redundant, the route column can be removed.

df.drop(columns=['Route'], inplace=True)
```

In [58]:
```
df
```

Out[58]:

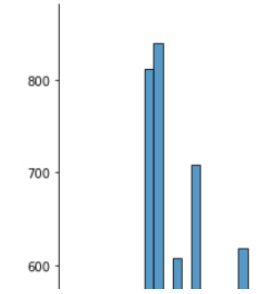
	Airline	Source	Destination	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min	Duration_hours	Duration_mins
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	02	50
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15	07	25
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25	19	00
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30	05	25
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35	04	45
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	Kolkata	Banglore	0	4107	9	4	19	55	22	25	02	30
10679	Air India	Kolkata	Banglore	0	4145	27	4	20	45	23	20	02	35
10680	Jet Airways	Banglore	Delhi	0	7229	27	4	8	20	11	20	03	00
10681	Vistara	Banglore	New Delhi	0	12648	1	3	11	30	14	10	02	40
10682	Air India	Delhi	Cochin	2	11753	9	5	10	55	19	15	08	20

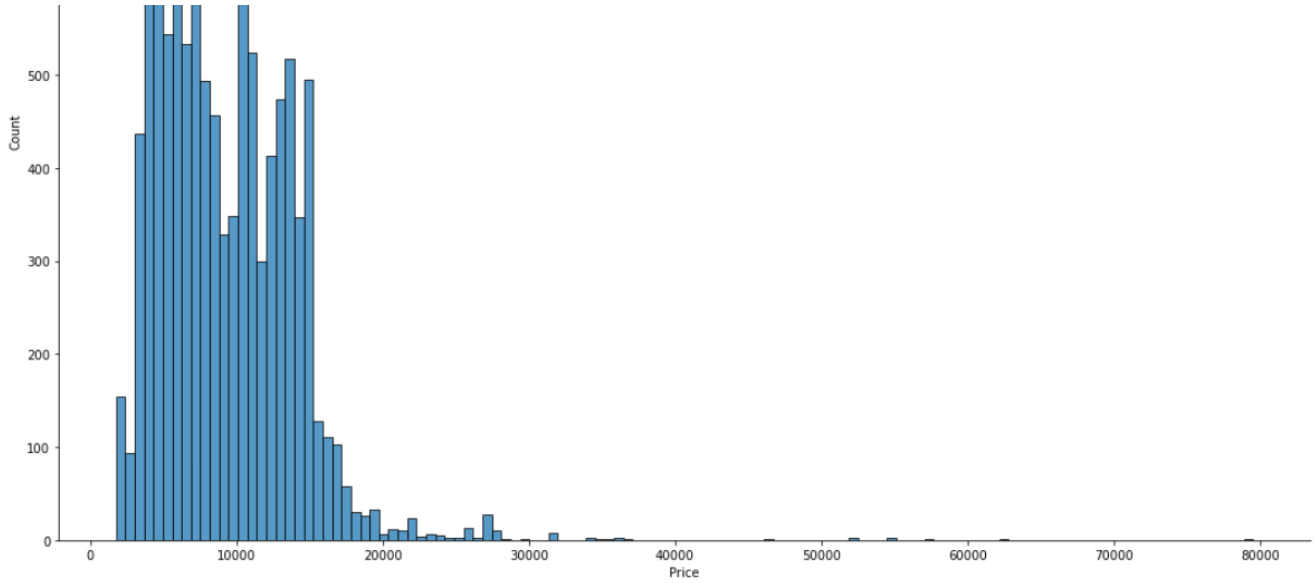
10683 rows × 13 columns

Price

In [59]:
```
sns.displot(df, x=df.Price, height=10, aspect=15/10)
# Visualizing the distribution of price


# The data is heavily distributed between 0 to 30,000.
```

Out[59]:
```
<seaborn.axisgrid.FacetGrid at 0x1ea89ee8b20>
```





Removing duplicate data

In [60]:
```
df.duplicated().value_counts()


# Total of 223 duplicated columns
```

Out[60]:
```
False    10460
True       223
dtype: int64
```

In [61]:
```
df.drop_duplicates(inplace=True)
```

In [62]:
```
# resetting the index after dropping 223 columns
df.reset_index(drop=True, inplace=True)
```

In [63]:
```
df.duplicated().value_counts()


# No any data is duplicated.
```

Out[63]:
```
False    10460
dtype: int64
```

In [64]:
```
df
```

Out[64]:

	Airline	Source	Destination	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min	Duration_hours	Duration_mins
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	02	50
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15	07	25
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25	19	00
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30	05	25
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35	04	45
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10455	Air Asia	Kolkata	Banglore	0	4107	9	4	19	55	22	25	02	30
10456	Air India	Kolkata	Banglore	0	4145	27	4	20	45	23	20	02	35
10457	Jet Airways	Banglore	Delhi	0	7229	27	4	8	20	11	20	03	00
10458	Vistara	Banglore	New Delhi	0	12648	1	3	11	30	14	10	02	40
10459	Air India	Delhi	Cochin	2	11753	9	5	10	55	19	15	08	20

10460 rows × 13 columns

Handling categorical data

The data in the airline column is nominal since the order of the data does not matter. The data is handled with one hot encoding.

In [65]:
```
df.Airline
```

Out[65]:
```
0            IndiGo
1         Air India
2       Jet Airways
3            IndiGo
4            IndiGo
```

```
10455       Air Asia
10456      Air India
10457    Jet Airways
10458        Vistara
10459      Air India
Name: Airline, Length: 10460, dtype: object
```

In [66]:

```
# Generating dummy variables and dropping the first to avoid the dummy variable trap
# with the column name using prefix

Airline = pd.get_dummies(df.Airline, drop_first=True, prefix='Airline')
Airline
```

Out[66]:

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vistara	Airline_Vistara Premium economy
0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
10455	0	0	0	0	0	0	0	0	0	0	0
10456	1	0	0	0	0	0	0	0	0	0	0
10457	0	0	0	1	0	0	0	0	0	0	0
10458	0	0	0	0	0	0	0	0	0	1	0
10459	1	0	0	0	0	0	0	0	0	0	0

10460 rows × 11 columns

In [67]:

```
df.Source.unique()
```

Out[67]:

```
array(['Banglore', 'Kolkata', 'Delhi', 'Chennai', 'Mumbai'], dtype=object)
```

In [68]:

```
Source = pd.get_dummies(df.Source, drop_first=True, prefix='Source')
Source
```

Out[68]:

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0
...	...	...	...	...
10455	0	0	1	0
10456	0	0	1	0
10457	0	0	0	0
10458	0	0	0	0
10459	0	1	0	0

10460 rows × 4 columns

In [69]:

```
# Similarly, with the destination column
Destination = pd.get_dummies(df.Destination, drop_first=True, prefix='Destination')
Destination
```

Out[69]:

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	1
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1
...	...	...	...	...	...
10455	0	0	0	0	0
10456	0	0	0	0	0
10457	0	1	0	0	0
10458	0	0	0	0	1

10459	1	0	0	0	0

10460 rows × 5 columns

In [70]:
```
# All the encoded columns are now appended into the dataset

df = pd.concat([df, Airline, Source, Destination], axis=1)
```

In [71]:
```
df
```

Out[71]:

	Airline	Source	Destination	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	...	Airline_Vistara Premium economy	Source_Chennai	Source_Delhi	Source_Kol
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	...	0	0	0	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	...	0	0	0	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	...	0	0	1	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	...	0	0	0	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10455	Air Asia	Kolkata	Banglore	0	4107	9	4	19	55	22	...	0	0	0	
10456	Air India	Kolkata	Banglore	0	4145	27	4	20	45	23	...	0	0	0	
10457	Jet Airways	Banglore	Delhi	0	7229	27	4	8	20	11	...	0	0	0	
10458	Vistara	Banglore	New Delhi	0	12648	1	3	11	30	14	...	0	0	0	
10459	Air India	Delhi	Cochin	2	11753	9	5	10	55	19	...	0	0	1	

10460 rows × 33 columns

In [72]:
```
df.drop(columns=['Airline', 'Source', 'Destination'], axis=1, inplace=True)
```

In [73]:
```
df
```

Out[73]:

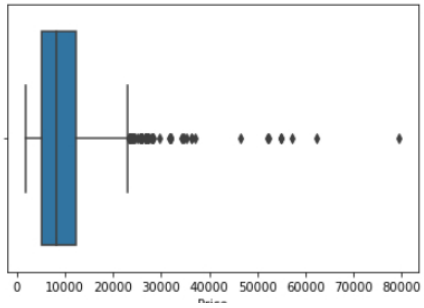
	Total_Stops	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min	Duration_hours	Duration_mins	...	Airline_Vistara Premium economy	Source_Chennai	Source_Dell
0	0	3897	24	3	22	20	1	10	02	50	...	0	0	
1	2	7662	1	5	5	50	13	15	07	25	...	0	0	
2	2	13882	9	6	9	25	4	25	19	00	...	0	0	
3	1	6218	12	5	18	5	23	30	05	25	...	0	0	
4	1	13302	1	3	16	50	21	35	04	45	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10455	0	4107	9	4	19	55	22	25	02	30	...	0	0	
10456	0	4145	27	4	20	45	23	20	02	35	...	0	0	
10457	0	7229	27	4	8	20	11	20	03	00	...	0	0	
10458	0	12648	1	3	11	30	14	10	02	40	...	0	0	
10459	2	11753	9	5	10	55	19	15	08	20	...	0	0	

10460 rows × 30 columns

Finding Outliers

In [74]:
```
sns.boxplot(x=df.Price)
```

Out[74]:
```
<AxesSubplot:xlabel='Price'>
```



Price

In [75]:
```
# Finding outliers in the price column using Z-score
price_arr = np.array(df.Price)
```

In [76]:
```
outliers_index = []

def detect_outliers(data):
    threshold = 3 #3 standard deviation
    mean = np.mean(data)
    std = np.std(data)

    for i,value in enumerate(data):
        z_score = (value-mean)/std
        if np.abs(z_score)>threshold:
            outliers_index.append(i)
    return outliers_index
```

In [77]:
```
outliers_index = detect_outliers(price_arr)
print(outliers_index)
```

```
[123, 396, 486, 510, 597, 628, 657, 784, 825, 935, 945, 958, 974, 1194, 1244, 1339, 1420, 1462, 1474, 1625, 1650, 1778, 1909, 2044, 2087, 2096, 2483, 2543,
2604, 2621, 2677, 2904, 3010, 3088, 3231, 3372, 3505, 3667, 3974, 4476, 4627, 4779, 4960, 5080, 5312, 5378, 5597, 5636, 5645, 5654, 5673, 5680, 5789, 5916,
6240, 6332, 6497, 6509, 6526, 6901, 7253, 7258, 7279, 7437, 7454, 7515, 7608, 7619, 7649, 7794, 7909, 7967, 8325, 8344, 8408, 8466, 8715, 8796, 8813, 8844,
8873, 9038, 9081, 9455, 9536, 9785, 9862, 9920, 9965, 9987, 9992, 10146, 10157, 10176, 10229, 10299]
```

In [78]:
```
df.iloc[123].Price
```

Out[78]:
```
27430
```

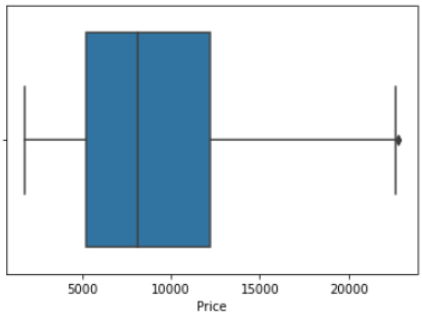
In [79]:
```
# Dropping all the outliers
df.drop(outliers_index, axis=0, inplace=True)
```

In [80]:
```
# Previous outlier data on the column has been deleted!

df.iloc[123].Price
```

Out[80]:
```
3540
```

In [81]:
```
sns.boxplot(x=df.Price)
```

Out[81]:
```
<AxesSubplot:xlabel='Price'>
```



Feature selection

In [82]:
```
df.columns
```

Out[82]:
```
Index(['Total_Stops', 'Price', 'Journey_Day', 'Journey_Month', 'Dep_Hour',
       'Dep_Min', 'Arrival_Hour', 'Arrival_Min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi'],
      dtype='object')
```

Rearranging the columns

In [83]:
```
X = df.loc[:,['Journey_Day', 'Journey_Month', 'Dep_Hour',
       'Dep_Min', 'Arrival_Hour', 'Arrival_Min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
```

```
        'Airline_Jet Airways', 'Airline_Jet Airways Business',
        'Airline_Multiple carriers',
        'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
        'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
        'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
        'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
        'Destination_Kolkata', 'Destination_New Delhi','Total_Stops']]
```

In [84]:
```
X.head()
```

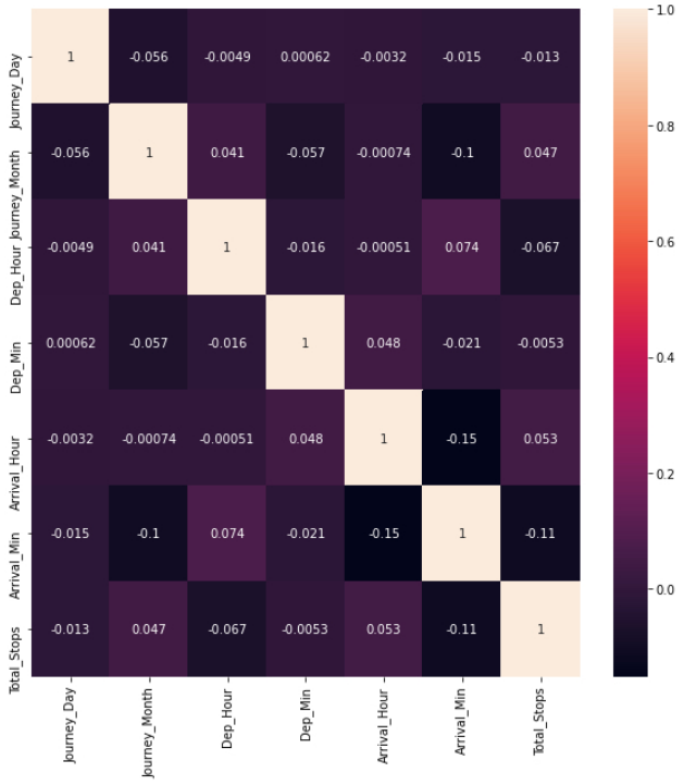
Out[84]:

	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	...	Source_Chennai	Source_Delhi	Source_Koll
0	24	3	22	20	1	10	02	50	0	0	...	0	0	
1	1	5	5	50	13	15	07	25	1	0	...	0	0	
2	9	6	9	25	4	25	19	00	0	0	...	0	1	
3	12	5	18	5	23	30	05	25	0	0	...	0	0	
4	1	3	16	50	21	35	04	45	0	0	...	0	0	

5 rows × 29 columns

In [85]:
```
X_prime = df.loc[:,['Journey_Day', 'Journey_Month', 'Dep_Hour',
        'Dep_Min', 'Arrival_Hour', 'Arrival_Min', 'Duration_hours',
        'Duration_mins','Total_Stops']]
```

In [86]:
```
# Checking correlation
plt.figure(figsize=(10, 10))
sns.heatmap(X_prime.corr(), annot=True)
plt.show()

# Highly correlated values can be dropped from here.
```



In [87]:
```
y = df.Price
```

Important feature selection using ExtraTreesRegressor

In [88]:
```
# Important feature selection using ExtraTreesRegressor

from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(X, y)
```

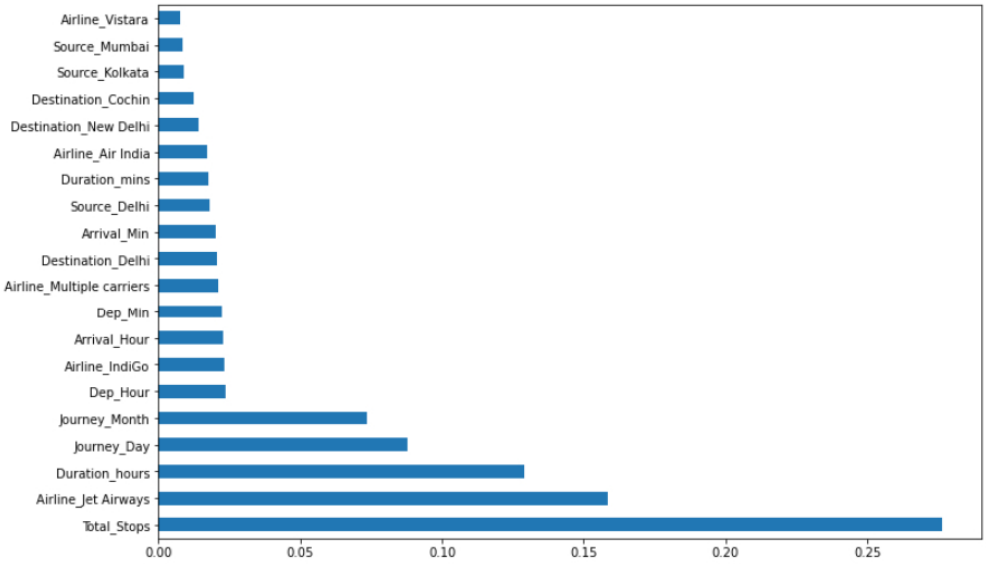
```
selection.fit(X, y)
```

Out[88]:
```
ExtraTreesRegressor()
```

In [89]:
```
# Trees regressor has found the important features

print(selection.feature_importances_)
```

```
[8.78185145e-02 7.35849230e-02 2.37950381e-02 2.24762465e-02
2.26031209e-02 2.01189099e-02 1.28997780e-01 1.76331302e-02
1.69378816e-02 2.86644085e-03 2.30367420e-02 1.58535280e-01
0.00000000e+00 2.11130167e-02 1.16916443e-03 4.99560456e-03
2.10936659e-04 7.49576846e-03 1.22665691e-04 7.61486181e-04
1.81365102e-02 8.71160432e-03 8.38598533e-03 1.25861698e-02
2.04255755e-02 6.26087481e-03 4.87097970e-04 1.42684060e-02
2.76465126e-01]
```

In [90]:
```
# Plotting the values for better visualization

plt.figure(figsize=(12, 8))
feature_imp = pd.Series(selection.feature_importances_, index=X.columns)
feature_imp.nlargest(20).plot(kind='barh')
plt.show()
```



Here, Total_Stops is the most important feature.

Profiling report

In [91]:
```
profile = ProfileReport(df, title="Airlines Profiling Report")
```

In [92]:
```
profile
```

Airlines Profiling Report

Overview | Variables | Interactions | Correlations | Missing values | Sample

Overview

Overview | Warnings 79 | Reproduction

Dataset statistics

Number of variables	31
Number of observations	10364
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0

Variable types

Numeric	9
Categorical	22

Duplicate rows (%)	0.0%
Total size in memory	1.1 MiB
Average record size in memory	108.0 B

Variables

df_index
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	10364	**Minimum**	0
Distinct (%)	100.0%	**Maximum**	10459

Out[92]:

Data Wrangling

Data wrangling is the process of transforming data from its original "raw" form into more readily used format. It prepares the data for analysis. It is also known as data cleaning, data remediation or data munging.

It can be both manual or an automated process. When the dataset is immense, the manual data wrangling is very tedious and needs automation.

It consists of six steps:

Step 1: Discovering In this step, data is understood more deeply. It is also known as the way of familiarizing with data so it can be further passed to different steps. During this phase, some patterns in data could be identified and the issues with the dataset is also known. The values which are unnecessary, missing or incomplete are identified for addressing.

Step 2: Structuring Raw data has a strong probability to be in a haphazard manner and unstructured and it needs to be restructured in a proper manner. Movement of data is made for easier computation as well as analysis.

Step 3: Cleaning In this step, data is cleaned for high quality analysis. Here, null values will have to be changed and formatting will also be standardized. It also includes the processes of deleting empty rows and removing outliers ensuring there are as less errors as possible.

Step 4: Enriching In this step, it is determined whether all the data necessary for a project is fulfilled. If not, the dataset is extended by merging with another dataset or simply incorporating values from other datasets. If the data is complete, the enriching part is optional. Once new datas are added from another dataset, steps of discovering, structuring, cleaning and enriching dataset needs to be repeated.

Step 5: Validating In this step, the states of the data (its consistancy and quality) are verified. If no issues are found to be resolved, the data is ready to be analysed.

Step 6: Publishing It is the final step where the data that has been validated is published. It can be published in different file formats ready for analysis by the organisation or an individual.

Sources: https://www.trifacta.com/data-wrangling/

https://online.hbs.edu/blog/post/data-wrangling

Data Cleaning

It is a process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated or improperly formatted.

When data from multiple dataset are combined, there is a strong possibility that the same data is replicated or even mislabeled. Using such data into the machine learning models are destined to give unexpected output.

Cleaning data is fairly simple and straight forward. There are four steps involved during the cleaning of data. Although the techniques may vary according to the types of data a company stores, the basic steps are common across all types of data.

Step 1: Removing duplicate and irrelevant observations

Data is collected thorugh different resources be it specific datasets, web scraping or any other medium. During the collection of data, some observations might repeat themselves due to a human error. Sometimes, the datasets might contain more information than we actually need. Such irrelevant as well as duplicate data needs to be dealt with in this step.

Step 2: Fixing structural errors

Data collected might have several structural issues including typos, incorrect data types, and weird naming convention which decreases the quality and reliability of data.

Step 3: Filtering unwanted outliers There might be some pieces of data that doesn't match the rest, called outliers. It can be due to improper data entry. But, that doesn't mean that every outlier is incorrect, its validity needs to be determined and if the outlier is proven to be irrelevant, it should be removed.

Step 4: Handling missing data The data with the removed outliers and some NaN fields are generally not accepted by many algorithms and that is why it should be handled. There are two ways to do so. The first one is dropping the entire observation. It is not optimal but necessary in some cases. Another option is to fill the missing data with observation and reference from other observations. It is also not ideal because the data is then based on assumptions and not the actual observation.

Step 5: Validating the data After all the above steps are perfomed, the data should be validated and the quality of data should be questioned. Not only that, it should be carefully observed whether the data brings any insights to light or not.

Sources: https://www.tableau.com/learn/articles/what-is-data-cleaning

Data Integration

It is a preprocessing method that involves merging of data from different sources in order to form a data store like data warehouse.

Issues in Data Integration:

1. Schema Integration and object matching

For example: Different datasets might contain same information with different labels which might not be visible at first glance.

1. Redundancy

For example: Different attributes might be redundant like if a dataset contains both the date of birth and the age details, age is redundant bacause it can simply be derived from date of birth.3

1. Detection and resolution of data value conflicts

For example: If a column in a dataset contains the price in NPR and another dataset contains the price details in USD, there might be conflicts in values if NPR data is simply converted to USD.

Sources: https://www.youtube.com/watch?v=UKUq7hZdZUw

Data Reduction

The data that the world is generating right now is tremendous. Data Reduction or Dimensionality Reduction helps to reduce p dimensions of the data into a subset of k dimensions where k<<n. The outcomes of doing this are the less computation/training time, less space required for storage, Better performance in most algorithms, removal of redundant features helping in multicollinearity and a lot more. There are many techniques for doing so including missing value ratio, low variance filter, high correlation filter, random forest, backward feature elimination, forward feature selection, factor analysis, Principal Component Analysis, Independent Component Analysis and so on.

Sources: https://www.analyticsvidhya.com/blog/2018/08/dimensionality-reduction-techniques-python/

Data Transformation

Data transformation is the process of changing the format, structure, or values of data. Due to data transformation, the data is easier for both humans and computers to use, data quality is tremendously improved, compatibility is facilitated between applications, systems, and types of data. However, there are a few challenges associated with data transformation i.e., the transformation can be expensive in terms of computing resources & licensing, lack of expertise can cause more problems than the problem it solves and the process can be resource-intensive.

Sources: https://www.stitchdata.com/resources/data-transformation/

Assigment

You are required to create a jupyter notebook that contains your results of undertaking the research detailed below. For submission, you should generate PDF/ipynb document. Make sure your PDF/ipynb has rendered correctly before submitting.

You must use the attached dataset. This is a data pre-processing and data preparation task in which you must understand and preprocess your data thoroughly and accurately.

1. Short Introduction about Data set
2. Understanding data and data pre-preprocessing
3. Extract derived features from data
4. Perform data pre-processing
5. Handle categorical data and feature encoding
6. Perform label encoding on data
7. Outlier Detection and Remove outlier from the dataset
8. Select the best feature using the feature selection technique

Note: Don't forget to explain each and every section of the solution as far as possible.

From the selected topic below write 350 words explaining how it could help with the data preprocessing.

1. Data Wrangling
2. Data Cleaning
3. Data integration
4. Data Reduction
5. Data Transformation

Your work needs to be fully referenced, which is good practice for your dissertation.

In [ ]: